ساختمان‌های داده و الگوریتم

تمرین چهارم - مرتب‌سازی و درهم‌سازی

محمد امانلو، فاطمه کرمی

تاریخ تحویل: ۱۴۰۲/۰۹/۲۶

۱. نمره

الگوریتمی طراحی کنید که با گرفتن یک آرایه که در آن هر عنصر حداکثر ۲۰ عنصر سمت چپ خود دارد که از او بزرگ‌تر باشند، این آرایه را در زمان $O(n)$ مرتب کند.

پاسخ :

می‌دانیم طبق این تعریف عنصر مینیمم آرایه قطعا در بین ۲۰ عنصر اول آرایه است پس با $O(۱)$ آن را پیدا کرده و با عنصر اول جایگزین می‌کنیم. سپس با تکرار همین الگوریتم به صورت بازگشتی باقی آرایه را هم مرتب می‌کنیم که زمان اجرای $O(n)$ دارد.

۲. نمره

یک الگوریتم با مرتبه زمانی $O(n \lg k)$ برای ادغام k لیست مرتب شده در یک لیست مرتب شده، که در آن n تعداد کل عناصر در همه لیست های ورودی است، ارائه دهید.

پاسخ :

از $head$ هر یک از k لیست‌ها یک $minheap$ بسازید. سپس، برای یافتن عنصر بعدی در آرایه مرتب شده، عنصر مینیمم را استخراج کنید (در زمان $O \lg(k)$). سپس، عنصر بعدی را از لیست کوتاه‌تری که عنصر استخراج‌شده در ابتدا از آن آمده است (در زمان $O(\lg(k))$) به هیپ اضافه کنید. از آنجایی که یافتن عنصر بعدی در لیست مرتب شده حداکثر زمان $O(\lg(k))$ طول می کشد، برای یافتن کل لیست، به $O(n \lg(k))$ مرحله نیاز دارید. .

۳. نمره

محمد فکر می‌کند که اگر روش زنجیره‌سازی مجزا $(separate\ chaining)$ برای ساخت یک $hash\ table$ را اینگونه تغییر دهد که در روش جدید هر لیست پیوندی در یک خانه آرایه به صورت مرتب شده باشد، نتیجتا به کارایی $(performance)$ بهتری خواهد رسید. در روش جدید محمد پیچیدگی زمانی برای جستجوی موفق، جستجوی ناموفق، درج و حذف را محاسبه کنید.

پاسخ :

هر دو نوع جستجوی موفق و ناموفق به زمان اجرای $\Theta(۱ + \lg(\alpha))$ نیاز دارند. درج و حذف مانند قبل به زمان اجرای $\Theta(۱ + \alpha)$ نیاز دارند چون پیچیدگی زمانی درج یا حذف از لیست مرتب شده خطی است.

۴. نمره

الگوریتم $Quick\ Sort$ شامل دو فراخوان بازگشتی به خود است. پس از آنکه $Quick\ Sort$ ، $Partition$ را فراخوانی می‌کند،

زیرآرایه سمت چپ را به صورت بازگشتی مرتب می‌کند و سپس زیرآرایه سمت راست را به صورت بازگشتی مرتب می‌کند. فراخوانی بازگشتی دوم در $Quick\ Sort$ واقعاً ضروری نیست و می‌توان با استفاده از یک ساختار کنترل تکرار شونده از آن اجتناب کرد. این تکنیک $tail\ recursion$ نامیده می‌شود. نسخه زیر از $Quick\ Sort$ را در نظر بگیرید که $tail\ recursion$ را شبیه سازی می‌کند:

```
TAIL-RECURSIVE-QUICKSORT(A, p, r)
1  while p < r
2      // Partition and sort left subarray.
3      q = PARTITION(A, p, r)
4      TAIL-RECURSIVE-QUICKSORT(A, p, q − 1)
5      p = q + 1
```

ثابت کنید که $TAIL-RECURSIVE-QUICKSORT(A, ۱, A.length)$ به درستی آرایه A را مرتب می کند.

پاسخ :

با استفاده از استقرا حکم را ثابت می‌کنیم. برای حالت پایه اگر A شامل ۱ عنصر باشد $p = r$ خواهد بود پس الگوریتم درجا تمام می‌شود. حال فرض کنید برای $۱ \leq k \leq n - ۱$ تابع مورد نظر آرایه A با k عنصر را به درستی مرتب می‌کند. حال برای آرایه A با n عنصر اگر q را محور قرار دهیم طبق فرض استقرا تابع مورد نظر زیر آرایه سمت چپ را که سایز اکیدا کوچک‌تری از A دارد به درستی مرتب می‌کند. سپس p به $q + ۱$ آپدیت می‌شود و به همین ترتیب زیرآرایه سمت راست مرتب می‌شود به طوری که انگار از ابتدا تابع به صورت $TAIL-RECURSIVE-QUICKSORT(A, \mathrm{q+۱}, \mathrm{n})$ فراخوانی شده باشد. این زیر آرایه هم سایز اکیدا کوچک‌تری از A دارد پس طبق فرض استقرا این زیر آرایه هم به درستی مرتب خواهد شد. در نتیجه کل آرایه به درستی مرتب شده‌است.

۵. نمره

الگوریتمی طراحی کنید که با گرفتن n عدد صحیح بین ۰ تا k پیش‌پردازشی روی ورودی انجام داده و سپس در زمان $O(۱)$ با گرفتن دو عدد a و b مشخص کند که چه تعداد از اعداد ورودی در بازه $[a, ..., b]$ هستند. پیش‌پردازشی که الگوریتم روی ورودی انجام می‌دهد باید با پیچیدگی زمانی $\Theta(n + k)$ باشد.

پاسخ :

ابتدا پیش پردازش را مانند الگوریتم $Counting\ Sort$ انجام می‌دهیم به صورتی که $C[i]$ نشان دهنده تعداد عناصر کمتر یا مساوی i در آرایه باشد. سپس با گرفتن اعداد a و b کافی‌ست مقدار $C[b]$ - $C[a - ۱]$ را محاسبه کنیم که پاسخ مساله است و زمان اجرای $O(۱)$ دارد.

۶. نمره

الگوریتمی با زمان اجرای $O(n)$ طراحی کنید که با گرفتن مجموعه S شامل n عدد یکتا و عدد مثبت $k \leq n$، k تا عددی را در S مشخص کند که نزدیک‌ترین به میانه S هستند.

پاسخ :

ابتدا در زمان خطی $\frac{k}{۲} - \frac{n}{۲}$ امین عدد بزرگ را در مجموعه پیدا می‌کنیم. سپس روی آن عنصر $Partition$ بندی می‌کنیم. در زیرآرایه بزرگ‌تری که توسط این محوربندی ساخته شده‌است، k امین بزرگترین عنصر را پیدا می‌کنیم و روی آن $Partition$ بندی می‌کنیم. عناصر درون زیرآرایه کوچک‌تری که توسط این محوربندی ساخته شده‌است، همان k عدد خواسته شده هستند.